

Научно- исследовательск ий практикум

СТАТИЧЕСКИЕ ПЕРЕМЕННЫЕ И ФУНКЦИИ.
СИНГЛТОН

Статические переменные

И ФУНКЦИИ. А КАК БЕЗ НИХ?

Ключевое слово `static`

`static` – это дополнительный модификатор переменной:

```
static int variable;
```

Этот модификатор показывает, что :

- переменная находится в текущей *единице трансляции*
- должна быть размещена в отдельной области памяти
- для неё используется всегда сначала используется *нулевая инициализация*
- создание такой переменной происходит всего один раз

Пример статических переменных

```
static int variable;

void demo()
{
    // Создадим статическую переменную в функции
    static int count = 0;
    // Теперь выведем её и увеличим значение
    cout << count << " ";

    count++;
}

int main()
{
    for (int i=0; i<5; i++)
        demo();
    cout << endl;
    cout << "Uninitialized variable " << variable << endl;
    return 0;
}
```

Starting /Users/amakashov/projects/build-s
0 1 2 3 4
Uninitialized variable 0
/Users/amakashov/projects/build-static_ele

Статические переменные в классе

Статическая переменная в классе – это переменная, «привязанная» не к конкретному объекту, а к классу в целом

Такие переменные характеризуют весь класс целиком

Для неё тоже используется нулевая инициализация

При этом внутри класса находится её *объявление*

Её *определение* должно быть в *единице трансляции*

Пример класса

Определение переменной
– в .cpp файле

```
// B smartpointer.h
class SmartPointer
{
public:
    SmartPointer();
    static int counter;

protected:
};

// B smartpointer.cpp
SmartPointer::SmartPointer()
{
}

int SmartPointer::counter;
```

Наш микро-main()

```
int main(int argc, char *argv[])  
{  
    cout << "Counter " << SmartPointer::counter << endl;  
    SmartPointer::counter = -1;  
    cout << "Counter " << SmartPointer::counter << endl;  
}
```

```
Starting /Users/amakashov/projects/build-static.  
Counter 0  
Counter -1  
/Users/amakashov/projects/build-static_elements.
```

Статические функции

Статические функции – это функции, выполняемые не для конкретного экземпляра, а для класса в целом

Очевидно, что такая функция НЕ МОЖЕТ иметь доступа к переменным класса - у неё ведь нет объекта

При этом такая функция может иметь доступ к статическим переменным класса

Применение статических функций

```
class SmartPointer
{
public:
    SmartPointer();
    static int GetCount();

protected:
    static int counter;
};

int SmartPointer::GetCount()
{
    return counter;
}
```

Реализация main

```
int main(int argc, char *argv[])  
{  
    cout << "Counter " << SmartPointer::GetCount() << endl;  
}
```

```
Starting /Users/amakashov/projects/build  
Counter 0  
/Users/amakashov/projects/build-static_e'
```

СИНГЛТОН

Синглтоном называют класс, который может существовать в программе всего в одном экземпляре

Такой класс должен создаваться специальным образом

Такое решение удобно, если у нас нужно по всей программе использовать один и тот же экземпляр класса для какой-либо задачи

Например, для документирования информации

Логгер

```
class Logger
{
public:
    Logger(const Logger&) = delete;
    Logger& operator = (const Logger&) = delete;

    std::ofstream& operator<< (std::string data);

protected:
    Logger();
    ~Logger();

    std::ofstream m_stream;
};
```

Его реализация

```
Logger::Logger()
```

```
{
```

```
    m_stream.open("out.txt");
```

```
}
```

```
Logger::~~Logger()
```

```
{
```

```
    m_stream.close();
```

```
}
```

```
std::ostream &Logger::operator<<(std::string data)
```

```
{
```

```
    cout << "Writing data to log" << endl;
```

```
    m_stream << data << endl;
```

```
    m_stream.flush();
```

```
    return m_stream;
```

```
}
```

Есть небольшая проблема

Да, наш класс невозможно скопировать

Но его невозможно и создать – ведь конструктор у нас `protected`

Как следствие, мы можем вызвать конструктор только изнутри класса

Как же выйти из этой ситуации?

Использовать статическую функцию!

Небольшая доработка

```
#include <fstream>

class Logger
{
public:
    Logger(const Logger&) = delete;
    Logger& operator = (const Logger&) = delete;

    std::ofstream& operator<< (std::string data);
    static Logger& Instance(); //ВОТ ОНО!!!

protected:
    Logger();
    ~Logger();

    std::ofstream m_stream;
};
```

Реализация

В данном случае реализация должна быть **СТРОГО** в .cpp - файле

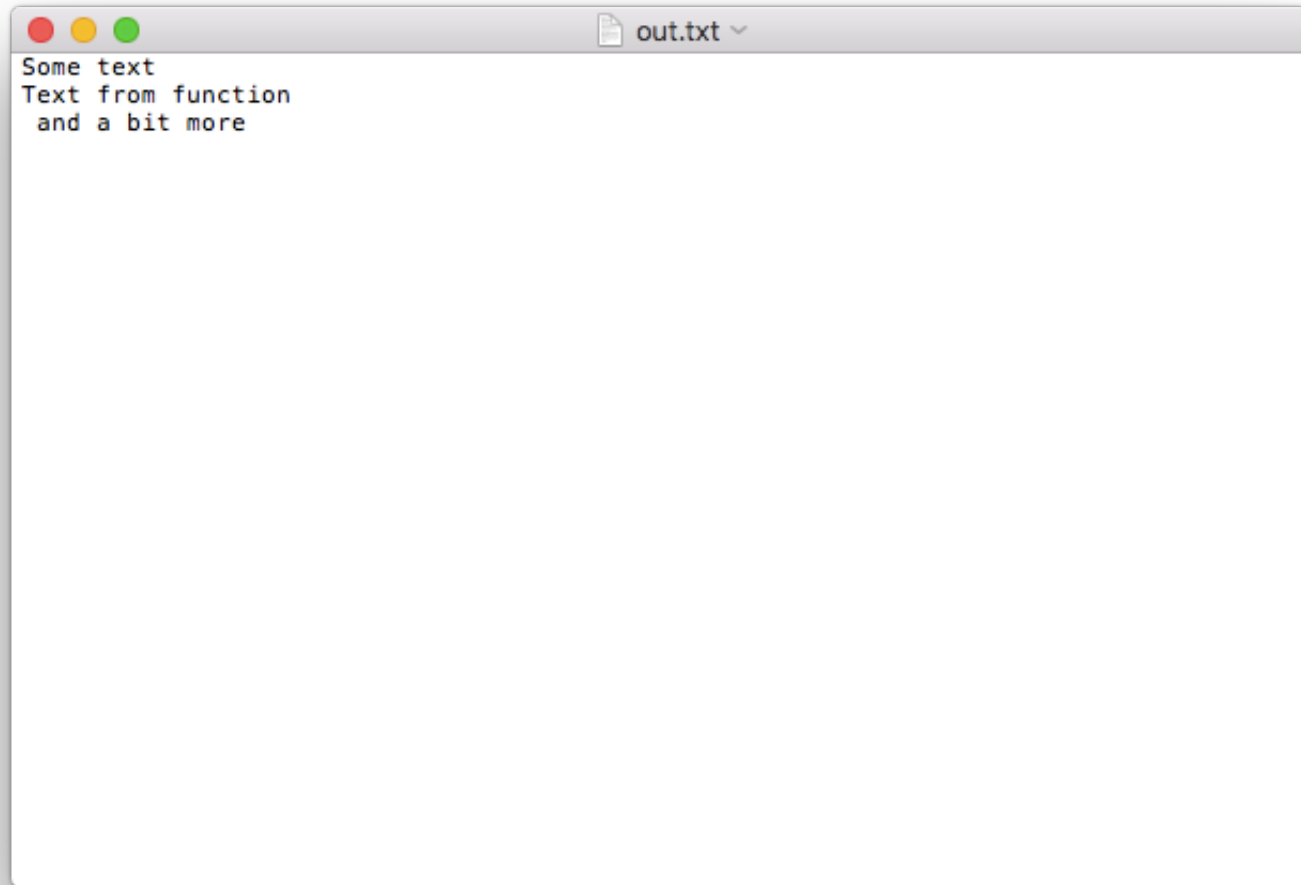
```
Logger &Logger::Instance()  
{  
    static Logger log;  
    return log;  
}
```


Как такой логгер вызвать

```
void SomeFunction()
{
    Logger& log = Logger::Instance();
    log<< "Text from function" << " and a bit more";
}

int main(int argc, char *argv[])
{
    Logger& log = Logger::Instance();
    log<< "Some text";
    SomeFunction();
}
```

Результат

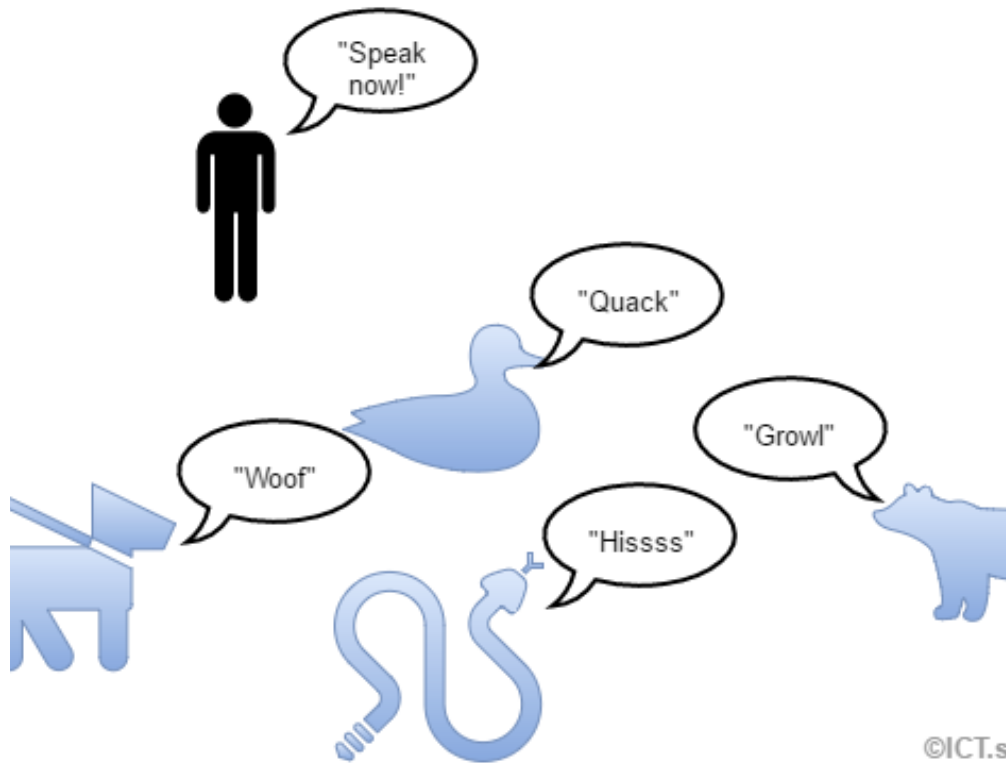


```
Some text
Text from function
and a bit more
```

Шаблоны

НЕСКОЛЬКО ВВОДНЫХ СЛОВ

Что такое полиморфизм?



Полиморфизм – средство объектно-ориентированных языков, позволяющее обрабатывать подобным образом объекты разных типов

Например, у нас есть несколько объектов с одинаковым методом `Speak()`

Статический и динамический полиморфизм

ДИНАМИЧЕСКИЙ

- Реализуется при выполнении программы (runtime)
- Способ - за счёт виртуальных функций
- Объявление в базовом классе метода виртуальным

СТАТИЧЕСКИЙ

- Реализуется при компиляции программы (compile time)
- Первый вариант реализации – перегрузка функций (его мы уже знаем)
- В частности, перегрузка операторов
- Второй вариант – использование шаблонов

Полиморфизм функций

```
int arraySumm (int* array, int arraySize)
{
    int summ (0);
    for (int i = 0; i < arraySize; i++)
        summ = summ + array[i];
    return summ;
}
```

```
double arraySumm (double* array, int arraySize)
{
    double summ (0);
    for (int i = 0; i < arraySize; i++)
        summ = summ + array[i];
    return summ;
}
```

И его применение

```
int main(int argc, char *argv[])
{
    double absPi = absValue (-M_PI);
    float absE = absValue (-M_E);
    std::cout << absPi << "\t" << absE << std::endl;

    int absTwo = absValue (-2);
    float absTree = absValue (-3);
    std::cout << absTwo << "\t" << absTree << std::endl;
    return 0;
}
```

Давайте вспомним, что же это такое...

```
Starting /Users/amakashov/projects/build-operator_ove  
Int summ 10  
Double summ 15.7  
/Users/amakashov/projects/build-operator_overload-Des
```


Как можно обойтись одной функцией?

- Мы можем использовать функцию шаблонного аргумента
- Шаблон – это средство языка, предназначенное для кодирования обобщённых алгоритмов, без привязки к конкретным типам
- Шаблоны могут использоваться в функциях, классах, функциях-членах класса и т.д.
- Синтаксис

```
template<typename T>
```

Или

```
template<class T>
```

Здесь T – это как раз шаблон

Пример шаблонной функции

```
template <typename T>
T arraySumm(T* array, int arraySize)
{
    T summs;
    for (int i=0; i<arraySize; i++)
        summ = summ +array[i];
    return summ;
}
```

main()

```
int main(int argc, char *argv[])
{
    int iArray[5] = {0,1,2,3,4};
    double dArray[5] = {3.14,3.14,3.14,3.14,3.14};
    cout << "Int summ " << arraySumm<int>(iArray, 5)
    << std::endl;
    cout << "Double summ " <<
    arraySumm<double>(dArray, 5) << std::endl;
    return 0;
}
```

Добавим
свой
класс...

```
class vector
{
public:
    vector(const double value = 0)
    {
        m_data[0] = m_data[1] = m_data[2] = value;
    }
    vector operator+(const vector& valVector)
    {
        ...
    }

    void Print() const
    {
        ...
    }
protected:
    double m_data[3];
};
```

И попробуем их сложить

```
int main(int argc, char *argv[])
{
    int iArray[5] = {0,1,2,3,4};
    double dArray[5] = {3.14,3.14,3.14,3.14,3.14};
    vector vArray[5] = {0, 1, 2, 3, 4};
    cout << "Int summ " << arraySumm<int>(iArray, 5) << std::endl;
    cout << "Double summ " << arraySumm<double>(dArray, 5) << std::endl;
    for (int i=0; i<5; i++)
        vArray[i].Print();
    vector summVec = arraySumm<vector>(vArray,5);
    cout << "Summ of vectors" << endl;
    summVec.Print();
    return 0;
}
```

Результат...

```
Starting /Users/amakashov/projects/build-operator_overl
Int summ 10
Double summ 15.7
0          0          0
1          1          1
2          2          2
3          3          3
4          4          4
Summ of vectors
10         10         10
/Users/amakashov/projects/build-operator_overload-Deskt
```

Попробуем массив из строк

```
int main(int argc, char *argv[])
{
    vector vArray[5] = {0, 1, 2, 3, 4};
    for (int i=0; i<5; i++)
        vArray[i].Print();
    vector summVec = arraySumm<vector>(vArray,5);
    cout << "Summ of vectors" << endl;
    summVec.Print();
    std::string strings[3] = {"String's", " summ ", "test"};
    std::string summ = arraySumm<std::string>(strings, 3);
    cout << summ << endl;
    return 0;
}
```

Результат

Starting /Users/amakashov/projects/build-operator_

0	0	0
---	---	---

1	1	1
---	---	---

2	2	2
---	---	---

3	3	3
---	---	---

4	4	4
---	---	---

Summ of vectors

10	10	10
----	----	----

String's summ test

/Users/amakashov/projects/build-operator_overload-

В каком случае это работает?

Для нашего T должны быть
определены операции
присваивания и сложения

```
template <typename T>
T arraySumm(T* array, int arraySize)
{
    T summs;
    for (int i=0; i<arraySize; i++)
        summs = summs + array[i];
    return summs;
}
```

Если чего-то не хватает, то...

```
../operator_overload/main.cpp:27:21: error: invalid operands to binary expression  
(‘vector’ and ‘vector’)
```

```
    summ = summ + array[i];
```

```
           ^~~~~~
```

```
../operator_overload/main.cpp:62:22: note: in instantiation of function template  
specialization ‘arraySumm<vector>’ requested here
```

```
    vector summVec = arraySumm<vector>(vArray,5);
```

```
           ^
```

Пример с
наследовани
ем —
похожий
результат

```
template <typename Who>
void Hello(Who& talker)
{
    talker.Greetings();
}

int main(int argc, char *argv[])
{
    Person person;
    Teacher teacher(64, "Genadiy");
    Student student(22, "Genady-jr.", 4);
    Assistant assistant(25, "Assistant",
3.5);

    Hello(person);
    Hello(teacher);
    Hello(student);
    Hello(assistant);

    return 0;
}
```

И ВЫВОД

```
Starting /Users/amakashov/projects/build-heir-Desktop-  
Hello, guys, my name is Anonymous  
Hello, students, let's start out lesson...  
Hello, I'm Genady-jr. and my average mark 4  
Hello, I'm Assistant and my average mark 3  
/Users/amakashov/projects/build-heir-Desktop-Debug/heir
```

Шаблоны в классах

```
template <typename Val>
class vector
{
public:
    vector(const Val value = 0)
    {
        m_data[0] = m_data[1] = m_data[2] =
value;
    }
    vector operator+(const vector& valVector)
    {
        vector ret;
        for (int i=0; i<3; i++)
            ret.m_data[i] = valVector.m_data[i]
+ m_data[i];
        return ret;
    }

    void Print() const
    {
        for (int i=0; i<3; i++)
            cout << m_data[i] << "\t";
        cout << endl;
    }

protected:
    Val m_data[3];
};
```

И их применение

```
int main(int argc, char *argv[])
{
    vector<double> vArray[5] = {0, 1, 2, 3, 4};
    for (int i=0; i<5; i++)
        vArray[i].Print();
    vector<double> summVec = arraySumm(vArray, 5);
    cout << "Summ of vectors" << endl;
    summVec.Print();
    return 0;
}
```

Шаблоны STL

Например

```
#include <vector>
```

```
std::vector<int> test_vector;
```

```
#include <list>
```

```
std::list<double> test_list;
```

cplusplus.com

cppreference.com